

Paradigmas de lenguajes de programación

Eduardo Bonelli

Departamento de Computación
FCEyN
UBA

“A language that doesn’t affect the way you think about programming, is not worth knowing”

Epigrams in Programming, Alan Perlis (primer ganador del Turing Award), ACM SIGPLAN, Sept. 1982

Marzo, 2006

Objetivos del curso

Conocer los pilares conceptuales sobre los cuales se erigen los lenguajes de programación de modo de poder

- Comparar lenguajes
- Seleccionar el más adecuado para una determinada tarea
- Prepararse para lenguajes/paradigmas futuros

Enfoque del curso

1 Angulo conceptual

- Introducción informal de conceptos de lenguajes de programación a través de ejemplos.

2 Angulo aplicado

- Implementación de los conceptos a través de intérpretes.

3 Angulo de fundamentos

- Estudio de aspectos formales ligados a algunos de los conceptos

Enfoque del curso

1 Angulo conceptual

- Introducción informal de conceptos de lenguajes de programación a través de ejemplos.

2 Angulo aplicado

- Implementación de los conceptos a través de intérpretes.

3 Angulo de fundamentos

- Estudio de aspectos formales ligados a algunos de los conceptos

Enfoque del curso

- 1 Angulo conceptual
 - Introducción informal de conceptos de lenguajes de programación a través de ejemplos.
- 2 Angulo aplicado
 - Implementación de los conceptos a través de intérpretes.
- 3 Angulo de fundamentos
 - Estudio de aspectos formales ligados a algunos de los conceptos

Algunos temas cubiertos - Angulo conceptual

- Inducción (variantes) y recursión, coinducción
- binding estático/dinámico, procedimientos de primera clase, recursión letrec, modalidades de pasaje de parámetros
- Asignación y efectos laterales
- Expresiones de tipo, sistema de tipos, chequeo de tipos, inferencia de tipos
- Resolución en lógica proposicional y de primer orden, Cláusulas de Horn, unificación, refutación, resolución SLD
- Objeto, clase, herencia, method dispatch estático/dinámico, polimorfismo, sistemas de tipos invariantes

Algunos temas cubiertos - Angulo aplicado

- Sintaxis, intérprete y type-checker para un lenguaje funcional con procedimientos, let-binding, recursión, referencias.
- Sintaxis, intérprete y type-checker para un lenguaje imperativo (lenguaje funcional con referencias).
- Sintaxis, intérprete y type-checker para un lenguaje orientado a objetos con clases, métodos, herencia simple, polimorfismo de subclase.

Algunos temas cubiertos - Angulo de fundamentos

- Paradigma funcional
 - PCF (Programmable computable functions): sintaxis (términos) y ejemplos
 - Sistema de tipos y semántica operacional
 - Equivalencia observacional (equ. obs) de términos, coinducción como técnica de prueba para demostrar equ. obs.
- Paradigma lógico
 - Resolución en lógica proposicional y de primer orden, Cláusulas de Horn, unificación, refutación, resolución SLD
- Paradigma orientado a objetos
 - Featherweight Java

Cátedra y modalidad

- Modalidad: Clases teóricas y prácticas

Teoría

- Eduardo Bonelli (ebonelli@dc.uba.ar)
- Martes: 17:00hs a 20:00hs

Práctica

- Analía Baño, Alejo Capparelli, Javier Altauz, Laura Lowenthal, Diego Figueira, Germán Gómez, Pablo Heiber, Guido de Caso
- Jueves: 19:30hs a 22:30hs

Recursos

Bibliografía

- Textos: no hay un texto principal, se utilizan varios, ver referencias en página web
- Apuntes: por ahora de
 - Inducción y recursión (autor: E.B)
 - Programación funcional (autor: Vega Weiss/Martínez-López)
- Papers
- Transparencias de teóricas

Página web

- Información al día del curso, consultar periódicamente

Recursos

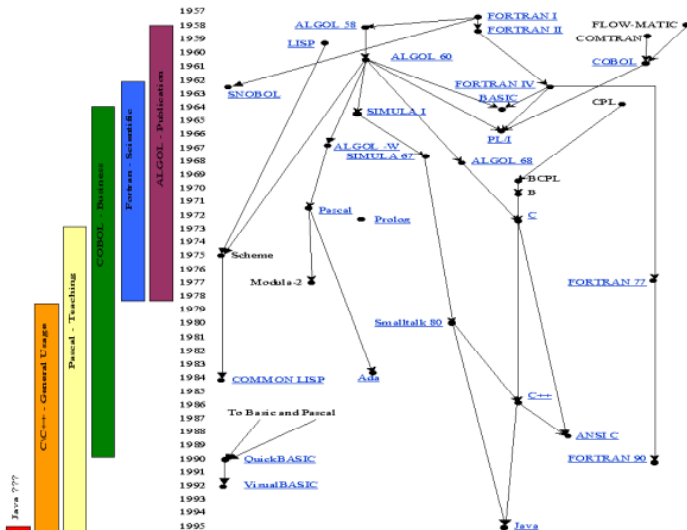
Software

- Implementaremos todos los intérpretes en Haskell
 - El intérprete (Hugs): <http://www.haskell.org>
 - Documentos (accesibles en el mismo sitio):
 - *A Gentle Introduction to Haskell*, Paul Hudak, John Peterson y Joseph H. Fasel.
 - *The Hugs 98 User Manual*, Mark P. Jones y John Peterson, 1999. (Manual del usuario; modestas 84 pginas)
 - *Haskell 98 Language and Libraries - The Revised Report*, Simon Peyton Jones (ed.), 2002. (La referencia definitiva sobre Hugs98, 277 pginas....)
- SWI-Prolog (programación lógica)
- Visualworks (Smalltalk - programación orientada a objetos)

Lenguajes de Programación

Lenguajes de Programación (LP)

- lenguaje usado para comunicar instrucciones a una computadora
- instrucciones describen *cómputos* que llevará a cabo la computadora
- noción de cómputo puede formalizarse (ej. máquinas de Turing, cálculo lambda, funciones recursivas, etc.): conjunto de funciones computables
- *computacionalmente completo* si puede expresar todas las funciones computables



Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
 - descripción del conjunto de secuencias de símbolos considerados como programas válidos
 - teoría de lenguajes formales bien desarrollada, comenzando a mediados de 1950 con Noam Chomsky como pionero
 - notación BNF (y EBNF) ampliamente utilizada; desarrollada por John Backus para Algol 58, modificada por Peter Naur para Algol 60
 - amplio abanico de herramientas para generar analizadores léxicos y parsers a partir de notación formal como BNF
- Sistema de Tipos
- Semántica

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
 - propósito: prevenir errores en tiempo de ejecución
 - en general, requiere anotaciones de tipo en el código fuente
 - ejemplos: evitar sumar booleanos, aplicar función a número incorrecto de argumentos.
 - análisis de tipos en tiempo de compilación: *chequeo de tipos estático*
 - análisis de tipos en tiempo de ejecución: *chequeo de tipos dinámico*
 - veremos en detalle en el Eje de Fundamentos
- Semántica

Definición de un Lenguaje

- Sintaxis
 - Sistema de Tipos
 - Semántica
- descripción del *significado* de instrucciones y expresiones
 - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
 - ¿Por qué semántica formal?^a:
 - Destructor británico H.M.S. Sheffield hundido en guerra de Malvinas. El radar de alerta de la nave estaba programado para identificar el misil Exocet como "aliado" debido a que el arsenal Inglés incluye el "homing device" de los Exocet y permitió que el misil alcanzara su blanco (el H.M.S. Sheffield).

Definición de un Lenguaje

- Sintaxis
 - Sistema de Tipos
 - Semántica
- descripción del *significado* de instrucciones y expresiones
 - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
 - ¿Por qué semántica formal?
 - Votos perdidos por computadora en Toronto. El distrito de Toronto finalmente abandonó votación electrónica.

Definición de un Lenguaje

- Sintaxis
 - Sistema de Tipos
 - Semántica
- descripción del *significado* de instrucciones y expresiones
 - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
 - ¿Por qué semántica formal?
 - 225 de los 254 pasajeros de Korean Airlines KAL 901 en Guam fallecen en accidente. Bug descubierto en altímetro barométrico del Ground Proximity Warning System (GPWS).

Definición de un Lenguaje

- Sintaxis
- Sistema de Tipos
- Semántica
 - descripción del *significado* de instrucciones y expresiones
 - puede ser informal (eg. Castellano) o formal (basado en técnicas matemáticas); semántica formal puede ser axiomática, operacional o denotacional
 - ¿Por qué semántica formal?
 - Falla en el despegue del satélite Ariane 5 causado por software error en la rutina de manejo de excepciones resultante de una mala conversión de un punto flotante de 64-bit a un entero.

Sintaxis - Ejemplo (BNF)

Ejemplo

```

$$\langle \text{Prog} \rangle ::= (\langle \text{Prog} \rangle \text{ union } \langle \text{Prog} \rangle) \mid ((\langle \text{Elem} \rangle \text{ into } \langle \text{Prog} \rangle) \mid \text{emptyset})$$

$$\langle \text{Elem} \rangle ::= a \mid b \mid c$$

```

Ejemplos de programas válidos

```
(a into emptyset), (b into (a into emptyset)),  
((a into emptyset) union emptyset)
```

Ejemplos de programas no válidos

```
(a into union), d union
```

Semántica - Ejemplo (denotacional)

- Fijar una estructura matemática. Ej. $E = \{A, B, C\}$ y $P = \wp(E)$
- Dar funciones de interpretación para cada no-terminal:

Funciones para $\llbracket \cdot \rrbracket_{\langle \text{Elem} \rangle}$ y $\llbracket \cdot \rrbracket_{\langle \text{Prog} \rangle}$

$$\llbracket a \rrbracket_{\langle \text{Elem} \rangle} = A$$

$$\llbracket b \rrbracket_{\langle \text{Elem} \rangle} = B$$

$$\llbracket c \rrbracket_{\langle \text{Elem} \rangle} = C$$

$$\llbracket (P \text{ union } Q) \rrbracket_{\langle \text{Prog} \rangle} = \llbracket P \rrbracket_{\langle \text{Prog} \rangle} \cup \llbracket Q \rrbracket_{\langle \text{Prog} \rangle}$$

$$\llbracket (E \text{ into } P) \rrbracket_{\langle \text{Prog} \rangle} = \{ \llbracket E \rrbracket_{\langle \text{Elem} \rangle} \} \cup \llbracket P \rrbracket_{\langle \text{Prog} \rangle}$$

$$\llbracket \text{emptyset} \rrbracket_{\langle \text{Prog} \rangle} = \emptyset$$

Semántica - Ejemplo (denotacional)

Ejemplo: Sea P el programa

$((a \text{ into emptyset}) \text{ union emptyset})$

Entonces

$$\llbracket P \rrbracket_{\langle \text{Prog} \rangle} = \{A\}$$

Donde (recordemos):

$$\llbracket a \rrbracket_{\langle \text{Elem} \rangle} = A$$

$$\llbracket b \rrbracket_{\langle \text{Elem} \rangle} = B$$

$$\llbracket c \rrbracket_{\langle \text{Elem} \rangle} = C$$

$$\llbracket (P \text{ union } Q) \rrbracket_{\langle \text{Prog} \rangle} = \llbracket P \rrbracket_{\langle \text{Prog} \rangle} \cup \llbracket Q \rrbracket_{\langle \text{Prog} \rangle}$$

$$\llbracket (E \text{ into } P) \rrbracket_{\langle \text{Prog} \rangle} = \{ \llbracket E \rrbracket_{\langle \text{Elem} \rangle} \} \cup \llbracket P \rrbracket_{\langle \text{Prog} \rangle}$$

$$\llbracket \text{emptyset} \rrbracket_{\langle \text{Prog} \rangle} = \emptyset$$

Procesadores de Lenguajes

Dos formas de ejecutar programas:

- 1 *interpretar*
- 2 *compilar*

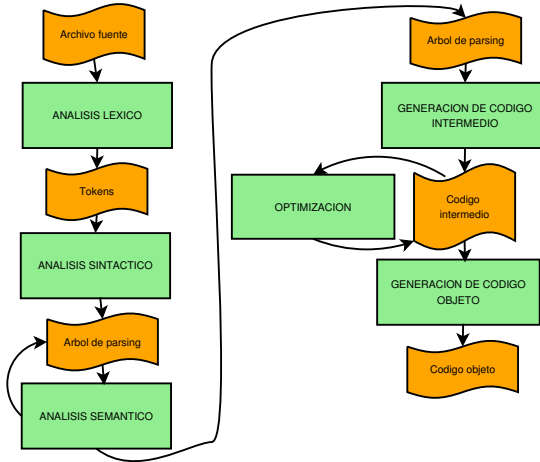
Intérprete

- programa que computa las acciones indicadas por un programa fuente
- es un *meta-programa* - programa que procesa otro programa
- utilizado para prototipar
- utilizado para dar semántica a un lenguaje
- nosotros lo utilizaremos para estudiar conceptos de LP

Compilador

- Traduce un programa *fuentes* en un programa *objeto*
- Muchas veces el programa fuente es un LP y el lenguaje objeto un lenguaje de máquina
- También es un *meta-programa*
- Parte de su tarea consiste en verificar que la cadena de entrada efectivamente se corresponde con un programa válido

Compilador



Compiladores vs Intérpretes

Ventajas de compiladores

- código compilado ejecuta (mucho) más rápido que código interpretado
- permite procesar módulos por separado

Ventajas de intérpretes

- programa fuente más pequeño que código compilado
- más fácil de escribir y modificar
- provee independencia de plataforma (portabilidad)

Paradigmas de Lenguajes

Diferentes maneras de expresar computación

- imperativo
- funcional
- lógico
- orientado a objetos

Otros: concurrente, dataflow, algebraico, etc.

Distinción a veces no está clara!

Veamos un ejemplo: calcular m^n ($n \geq 0$)

Imperativo

```
result := 1;  
while n>0 do  
    result := result * m;  
    n := n - 1;  
end while
```

Evaluación

- computación expresada a través de modificación reiterada de memoria *implícita*
- variables como abstracción de celdas de memoria
- resultados intermedios se almacenan en la memoria
- control basado en iteración

Funcional

$\text{power } m \ 0 = 1$

$\text{power } m \ (n+1) = m * \text{power } m \ n$

Evaluación

- computación expresada a través de la aplicación y composición de funciones
- no hay una memoria implícita
- resultados intermedios (salida de las funciones) son pasados directamente a otras funciones
- control basado en recursión

Lógico

```
power(m,0,1).  
power(m,n,result)  
  <- minus(n,1,n_sub1),  
    power(m,n_sub1,temp_result),  
    times(m,temp_result,result).
```

Evaluación

- computación expresada a través de proof search o alternativamente, por definición de predicados recursivos
- no hay memoria implícita
- resultados intermedios son pasados a través de unificación
- control basado en recursión

Orientado a Objetos

```
class Numero
  | val ... |
  instance method
    Valor          Power(pot)
    return val     if pot=0 then return 1
                  else return
                    ((send self Valor) *
                     (send self Power(pot-1)))
```

Evaluación

- computación a través del intercambio de mensajes e/objetos
- objetos se agrupan en clases, clases se agrupan en jerarquías
- resultados son pasados como parámetros a mensajes

Top five: “Great Works in Programming Languages”, B.Pierce

- C.A.R. Hoare. [An axiomatic basis for computer programming](#). Communications of the ACM, 12(10):576-580 and 583, October 1969.
- Peter J. Landin. [The next 700 programming languages](#). Communications of the ACM, 9(3):157-166, March 1966.
- Robin Milner. [A theory of type polymorphism in programming](#). Journal of Computer and System Sciences, 17:348-375, August 1978.
- Gordon Plotkin. [Call-by-name, call-by-value, and the \$\lambda\$ -calculus](#). Theoretical Computer Science, 1:125-159, 1975.
- John C. Reynolds. [Towards a theory of type structure](#). In Colloque sur la Programmation, Paris, France, volume 19 of Lecture Notes in Computer Science, pages 408-425. Springer-Verlag, 1974.

Vanguardia

Conferencias Europeas

- The European Joint Conferences on Theory and Practice of Software (ETAPS)
 - Foundations of Software Science and Computation Structures (FOSSACS)
 - Fundamental Approaches to Software Engineering (FASE)
 - European Symposium on Programming (ESOP)
 - International Conference on Compiler Construction (CC)
 - Tools and Algorithms for the Construction and Analysis of Systems (TACAS)
- International Colloquium on Automata, Languages and Programming (ICALP)
- Computer Science Logic

Vanguardia

Conferencias ACM

- Principles of Programming Languages (POPL)
- International Conference on Functional Programming (ICFP)
- Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)
- Programming Language Design and Implementation (PLDI)
- Principles and Practice of Parallel Programming (PPOPP)